

Software Architecture Concepts and Views UML Introduction

Tilt Thompkins
Director, Middleware Technologies

DR&CG Meeting
March 24, 2003

Outline And Biases

- **Biases - Importance of**
 - Interface-based design
 - Good programmers vs. Languages
 - Iterative development of requirements and design
- **Outline - TENA free discussion of**
 - Software architecture concepts
 - Visio Demo
 - UML tutorial
- **Be happy to discuss our TENA experiences and plans at the end of the day**
 - TENA - RMI alternative functionality - JXTA

How Did Software Architecture Get To Be An Important Idea Anyway

- **Interesting characteristics of modern systems**
 - **Homogenous hardware/software configurations, constantly evolving**
 - **Remote autonomous processing**
 - **Distributed, replicated, non-uniform state - time**
 - **Asynchronous, insecure, variable speed communications**
 - **Frequent partial failures**
- **Difficult issues to grapple with and not particularly connected to any specific application**
- **System development will be much easier if there is only one solution to these issues**

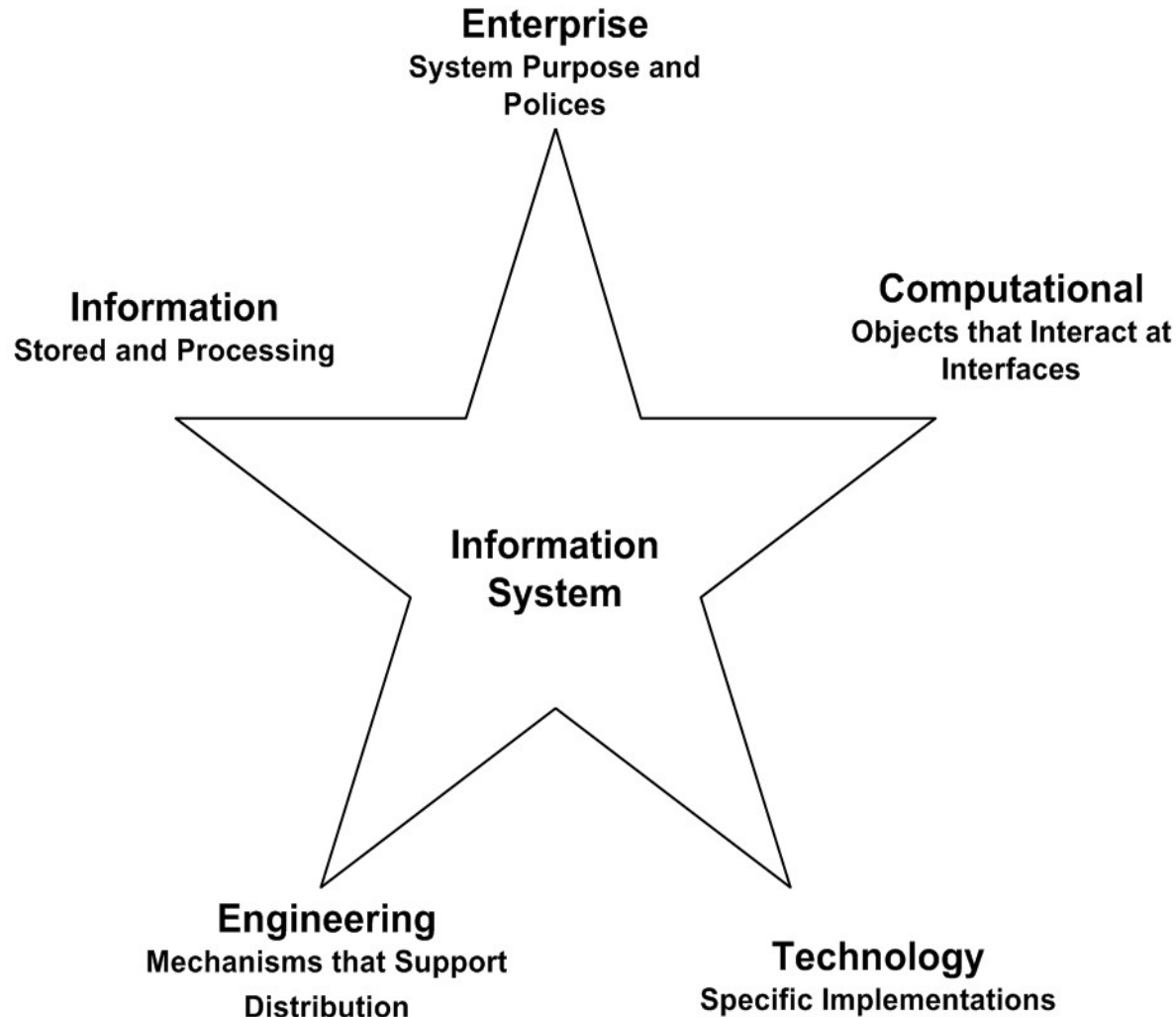
- **Without doubt the most ill-defined, important concept I know about.**
- **The most useful definition I can give you is:**
- **Rules by which the non-functional requirements are achieved.**
 - **Requirements other than computing correct results.**
 - **Constraints.**
 - » **Apply to the systems a whole.**
 - e.g., Inter-operability.
 - **Qualities.**
 - » **Usability, extensibility.**
 - » **QOS, performance, reliability.**

- **Scale**
 - The amount of simultaneous computing and distribution a system must support
- **Security**
 - Protection from unauthorized users
 - Protection from unauthorized uses
- **Fault-tolerance**
 - Up-time
 - Response to failures of components

The Battlegrounds For Software Architecture Consultants Are:

- **Terminology**
 - Solvable
- **A-Priori views**
 - Solvable by rational people
 - Discuss
 - » RM-ODP: Reference Model for Open Distributed Processing
 - » An HP approach
 - » Favorite business consultant view
- **Completeness: CMM vs. Extreme Programming**
 - Religious debate
 - Don't go there
 - Value, truth, and beauty in both camps

RM-ODP: Reference Model For Open Distributed Systems

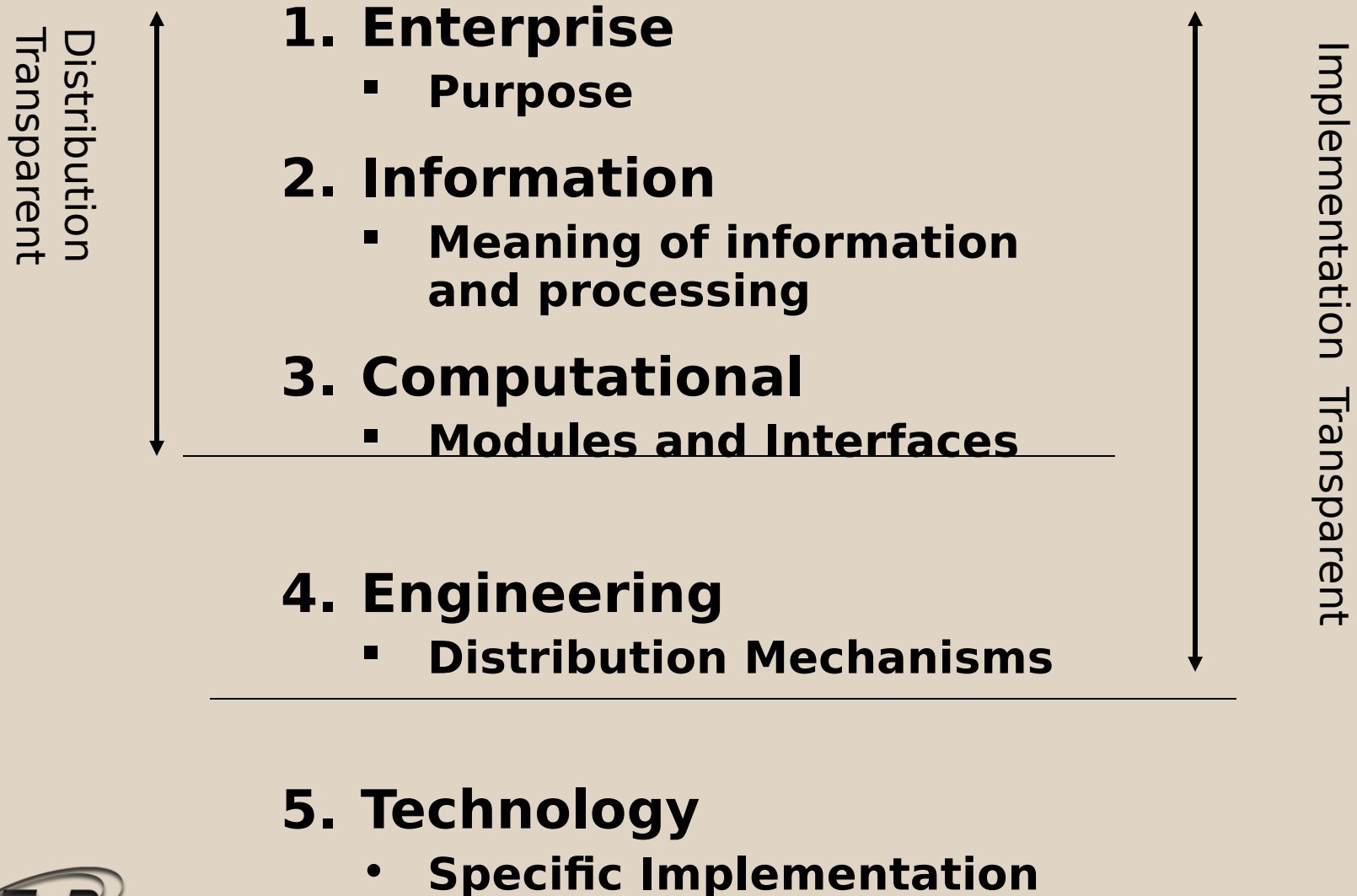


Open Distributed Processing Object Model

- **Objects are entities containing information and offering services**
 - Arbitrary granularity, behavior, and parallelism
- **System is interacting objects**
 - Interactions between objects are not constrained
- **Encapsulation**
 - Information hidden and accessible only through interfaces
 - No side effects
- **Abstraction**
 - Processing implementation hidden behind interfaces

RM-ODP Viewpoints

Design Independence



RM-ODP Viewpoints

Software Engineering

Requirements
Analysis

Functional
Specification

Design

Implementation

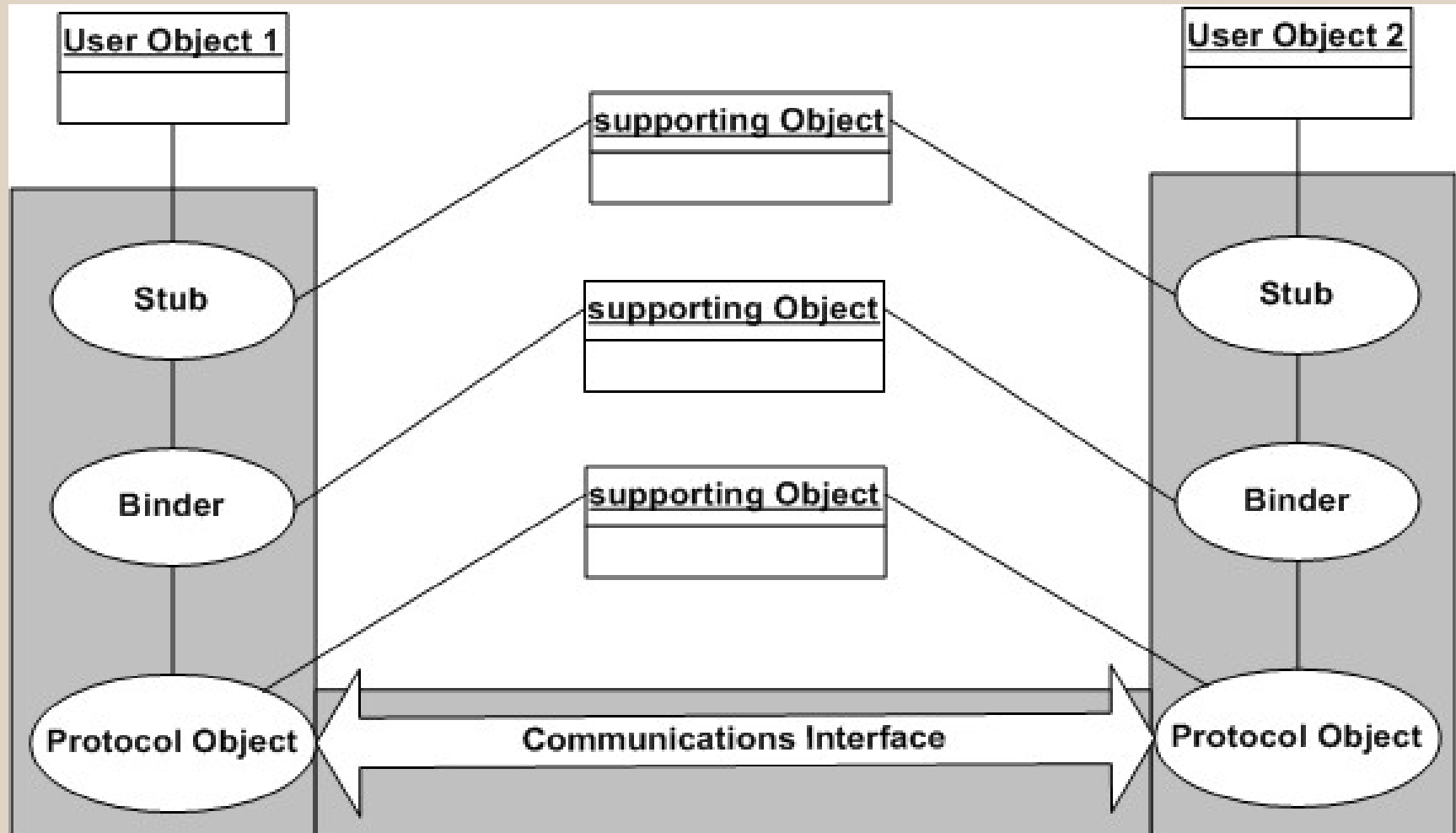
Enterprise

Information
Computational

Engineering

Technology

Channel Abstraction



- **Functionality which hides some common complexity**
- **Intended to shift effort from application developer to infrastructure developer**
- **Recipes for common-case solutions**
 - **Optional don't force for all applications**

Transparency Types

| Transparency | Guarantee |
|--------------|--|
| Access | Masks platform-protocol difference in representation and invocation mechanisms |
| Failure | Masks failures and recover of Objects |
| Location | Masks information needed to find and bind |
| Migration | Masks awareness of changes in location of the object from itself |
| Relocation | Masks changes in location of interfaces from client objects |
| Replication | Maintains consisting of a group of replica objects with a common interfold |
| Persistence | Masks activation and deactivation |
| Transactions | Masks coordination to achieve consistency |

CORBA provides support for location, access, and partial support for persistence. Multi-language support a key issue.

Access transparency is achieved by configuring the channel with stubs which:

- **Are accessed with local procedure call semantics**
- **Convert the interaction into a sequence of message passing**
- **Marshals and unmarshals data to convert between different representations**

Relocation Transparency

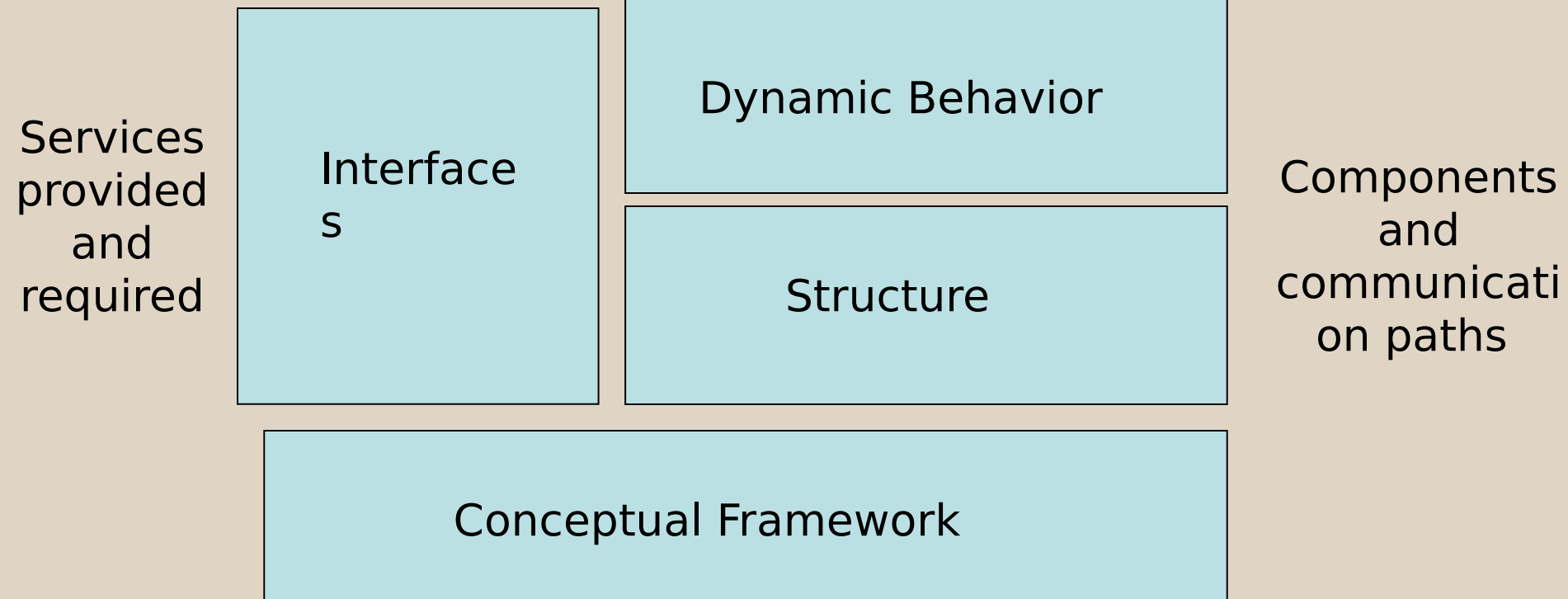
Achieved by configuring the channel with binder objects which:

- **Reports the location of it's processing thread to a locator service when the object is created, deleted, or moved**
- **Obtains the location of any interacting object**
- **Consequently**
 - » **Interacting user objects need not know each other's location**
 - » **Either user object can be moved**

- **Transactional transparency is very difficult to provide**
- **Requires**
 - **Reporting of the execution/undo of actions of interest to a transaction service**
 - » **Reading/writing transaction protected data**
 - **Interaction with the transaction service to arrange commit/abort behavior**
 - **Implement/start undo actions and restart transactions on request of the transaction service**

ODP Too Abstract For Real Designs: An HP Approach

How the components interact and change state



Meta-Architecture

- **Vision**
- **Style and philosophy**
- **Key Concepts and Mechanisms**

Architecture

- **Structure and Relationship.**
- **Interface definition**
- **Static and dynamic views**

Architecture Guidelines and Policies

- **Standards**
- **Frameworks**

- **Batch Sequential**
- **Pipes and Filters**
- **Main Program and Subroutines**
- **Object-Oriented**
- **Layered - Call and Return**
- **Communicating process and event systems**
- **Repository and Blackboard**
- **Rule-Based**
- **Communication Middleware**

- **Conceptual**
 - **Components and connections**
 - » **CRC cards level**
 - **Component responsibility collaborators rational**
- **Logical**
 - **Structure - interface definitions- protocols**
- **Execution**
 - **Maps to processes aid tasks**
 - **Synchronization requirements**

Paradigms: Background For UML

Discussions

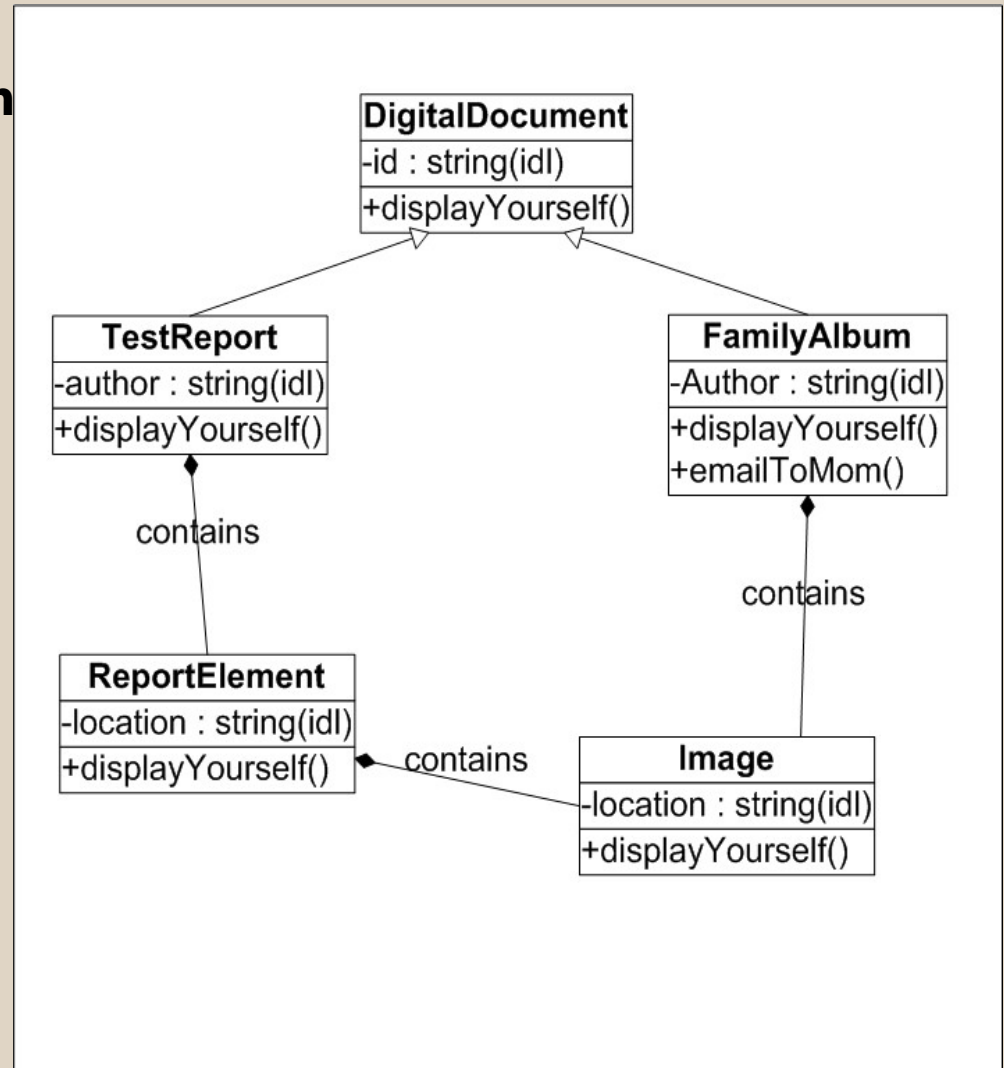
- **Procedural**
- **Object-Oriented**
- **Components**

Procedural

- **Common definition is that**
 - **Computer program stores algorithms separate from data**
 - **Algorithms retrieve, transform, and store data**
- **Weakness is that a data change may ripple through an entire program**

Object-Oriented

- **Key idea - Encapsulation**
 - Store data and algorithm to manipulate data together
 - Hide implementation details behind Interface
- **Key language concepts:**
 - Identity
 - Classification
 - Polymorphism
 - Inheritance



O-O Heresies from Dr. Thompkins

- **Encapsulation and Polymorphism are properties of interface design, not programming languages**
- **Inheritance is certainly one way to generate reuse, but interface design is probably the first order effect**
- **Essentially all complex software constructs are only modifiable by their original developers, regardless of the language they are written in.**
 - **Interface design is the key way to evolve systems that utilize multiple constructs**

Components

- **Key principles**
 - **Encapsulation - Polymorphism**
 - **Late Binding**
- **Focus reuse efforts on composition at the interface design level not inheritance**
- **Late-Binding gives interoperability across languages and systems**
- **Component Infrastructures**
 - **Sun and J2E2**
 - **Microsoft and .Net**
 - **CORBA**
 - **HLA and TENA ???**

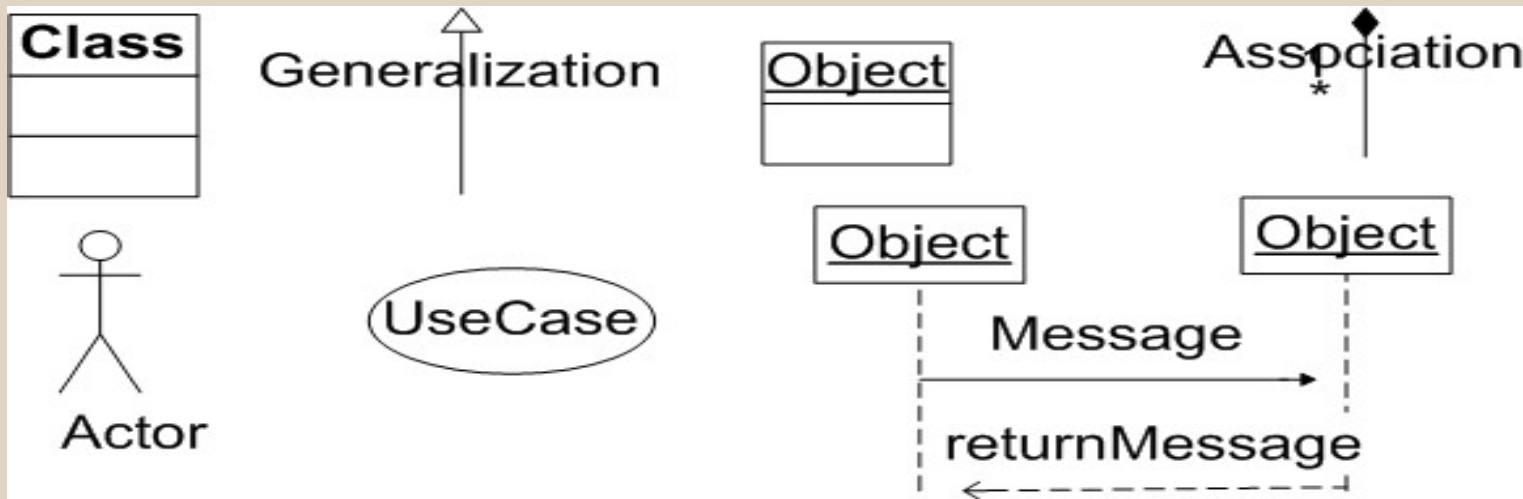
Comments On The Object-Model Approval Task

- **Group has been tasked to review some detailed models developed over time by the technical specialists**
 - **DR&CG needs to be clear on what value it plan to add to that process before we “design” the process.**
- **I believe there needs to be additional focus at levels of abstraction above the Object Models**
- **Some examples:**
 - **What are the common usage scenarios**
 - **What qualities must an infrastructure provide to support those scenarios**
 - **Build some example applications that can be distributed to potential users**

Break And Questions

Software Architecture Descriptions

UML: Unified Modeling Language

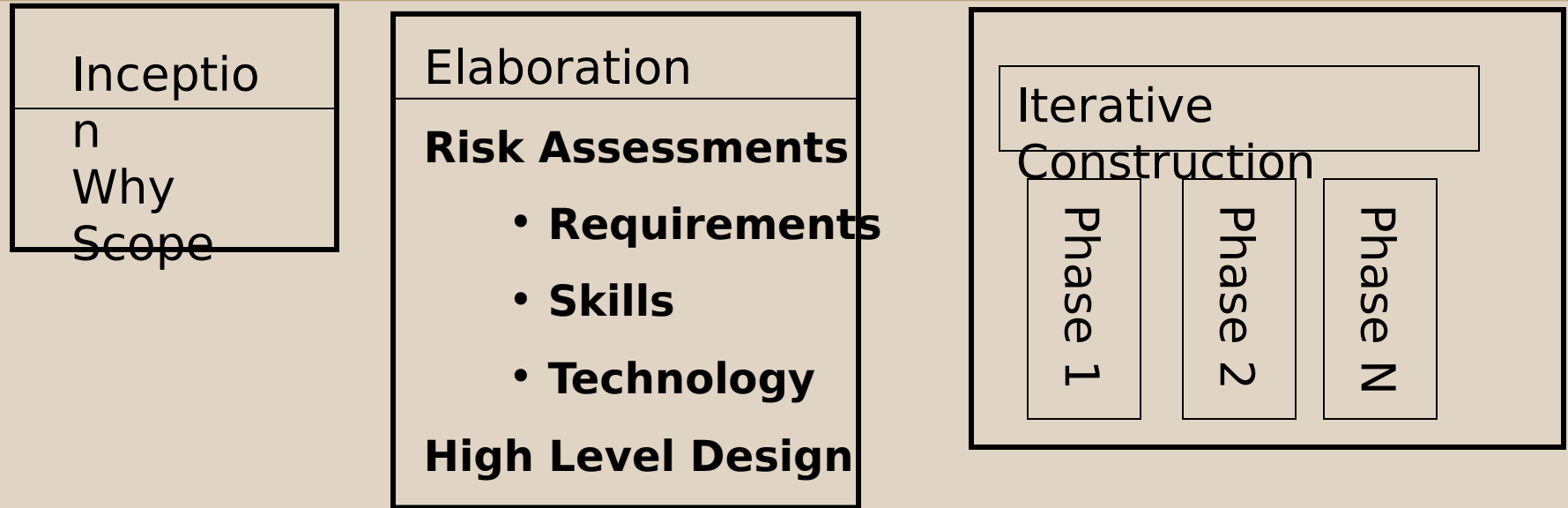


More that 100 symbols at my last
count

The issue is what/when to use:

- Development process view
- Architecture view

Development Process View



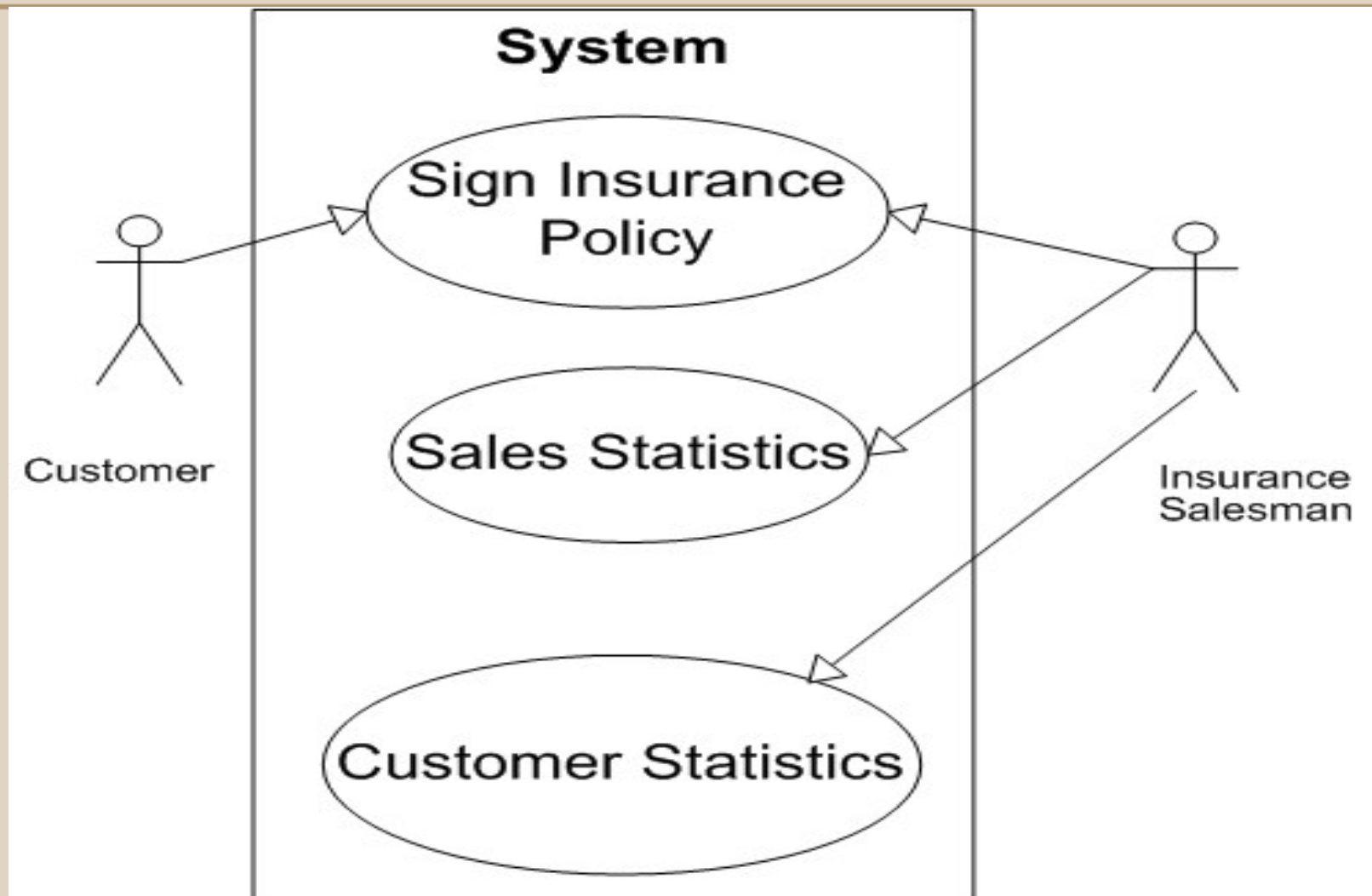
UML Tools

- Use Cases
- Context Diagrams
- Class Diagrams
- Activity

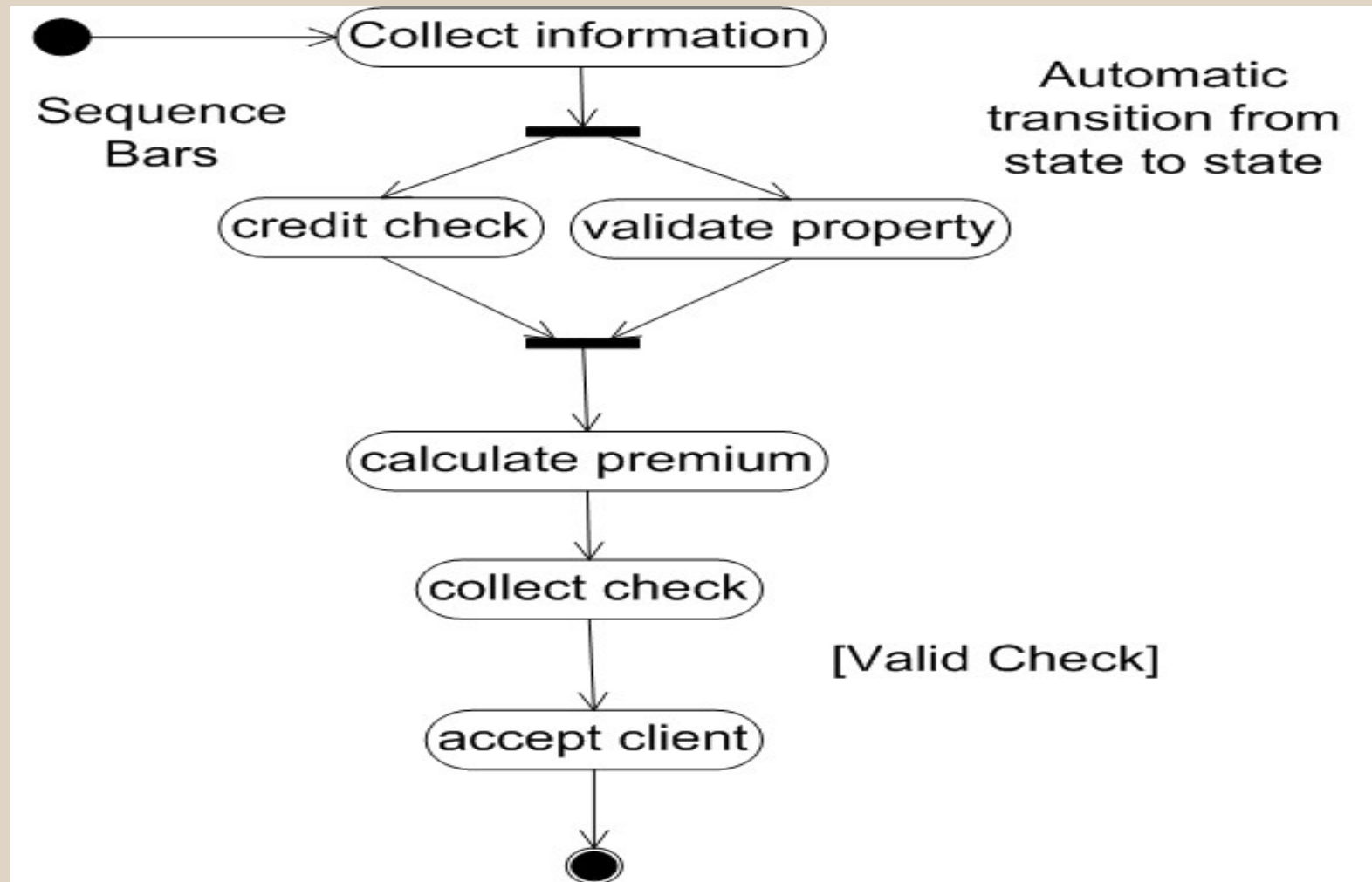
UML Tools

- Component Diagrams
- State Charts
- Sequence Diagrams

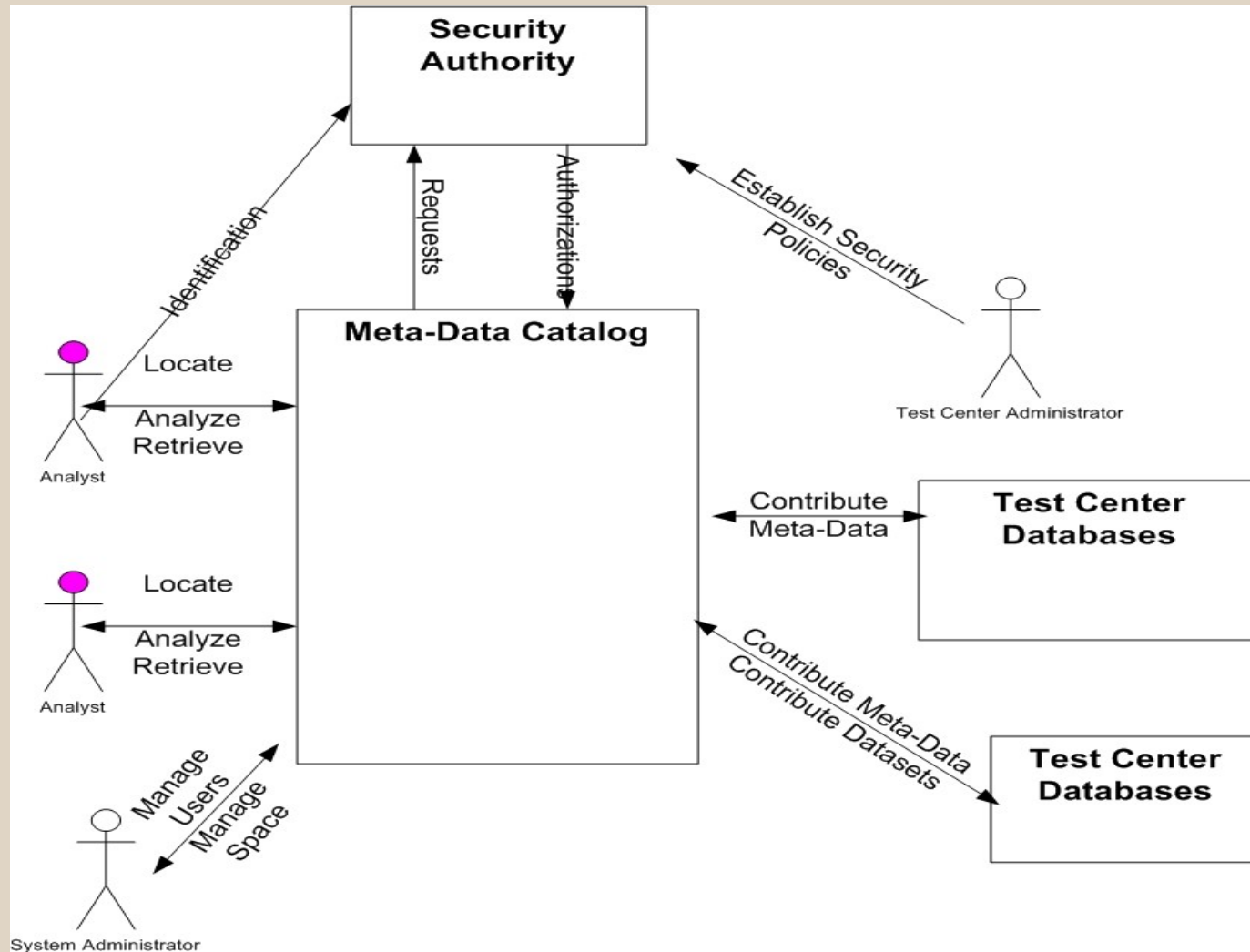
Context diagram



Activity Diagrams



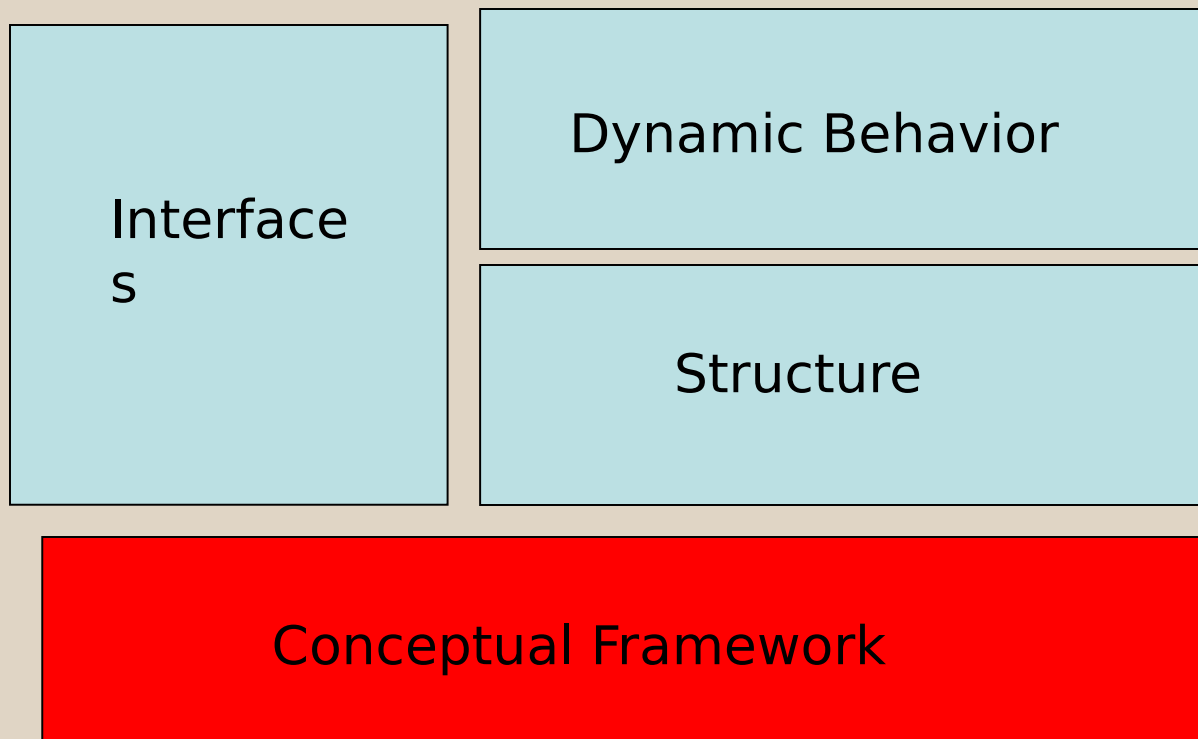
More Relevant Context Diagram



Use Cases

- **Describes things the system does to provide to provide value to the Actors**
- **Usually text that focus on key success paths**
 - **Failures tackled as separate Use Cases**
- **Typical Components**
 - **Name**
 - **Assumptions**
 - **Pre-conditions**
 - **Dialog**
 - **Post - conditions**
 - **Exceptions**
 - **Future Enhancements**
 - **Issues**

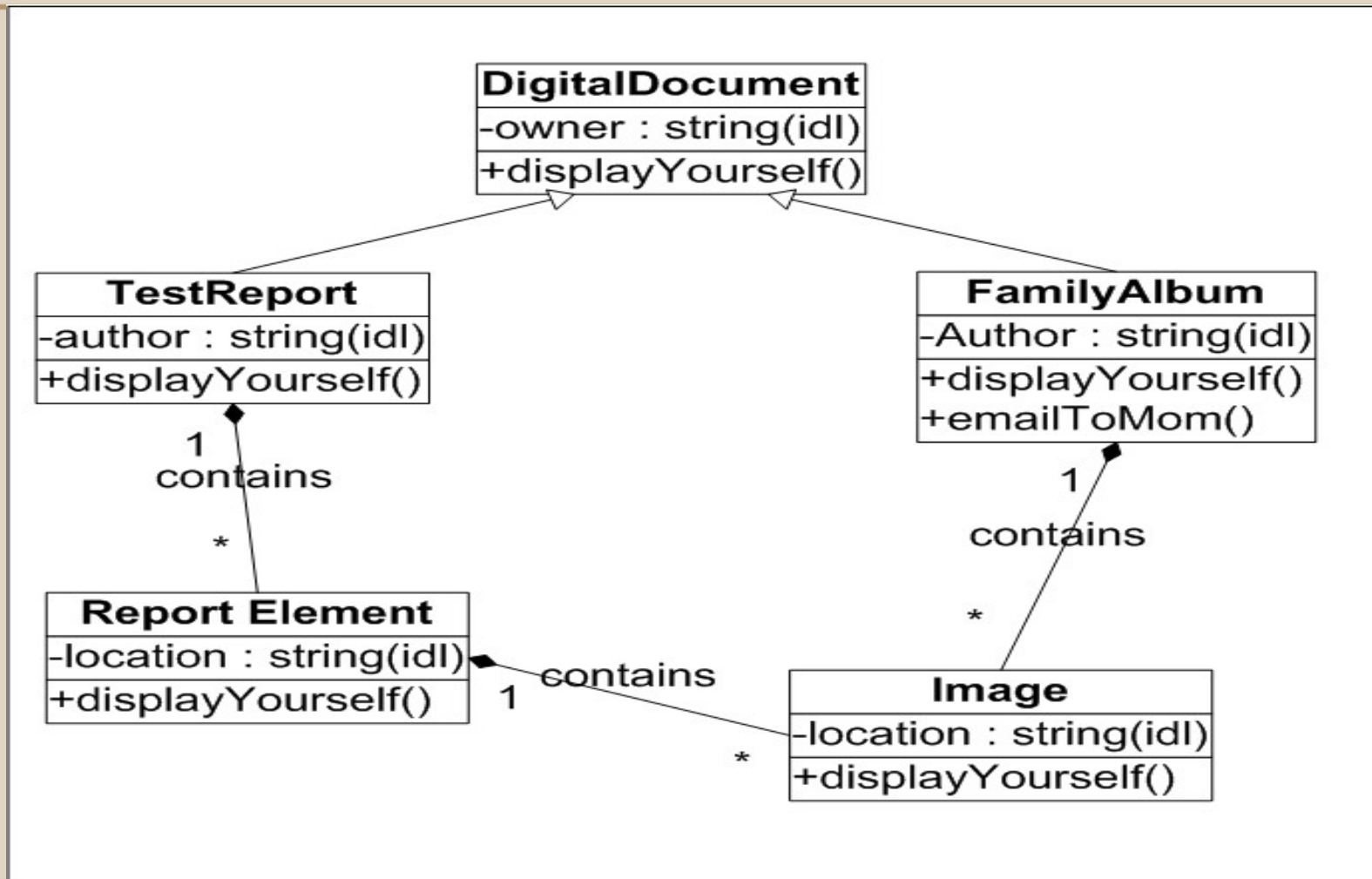
Using UML For Software Architecture Representations



Class Diagrams

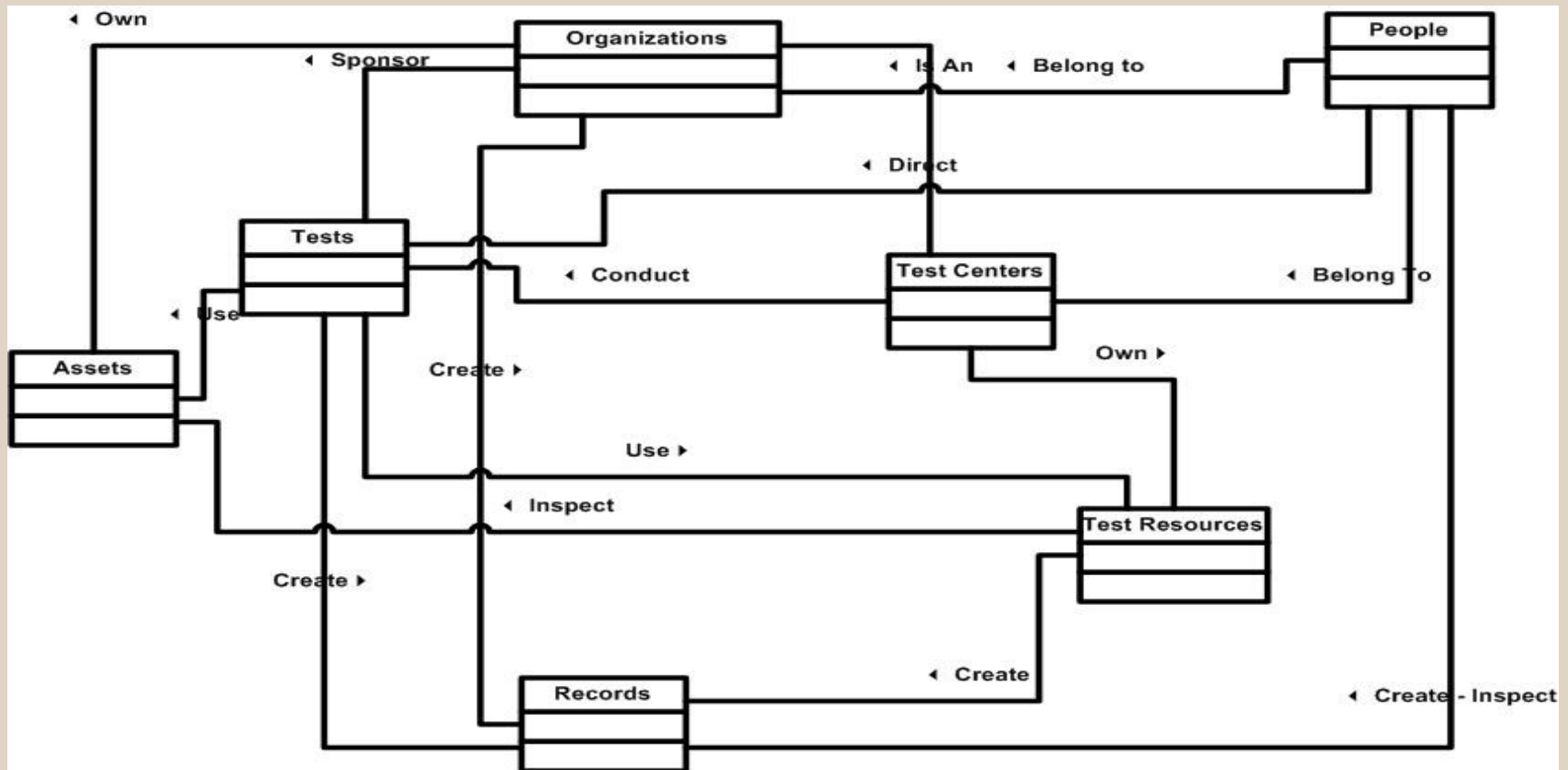
- application domain
- software design

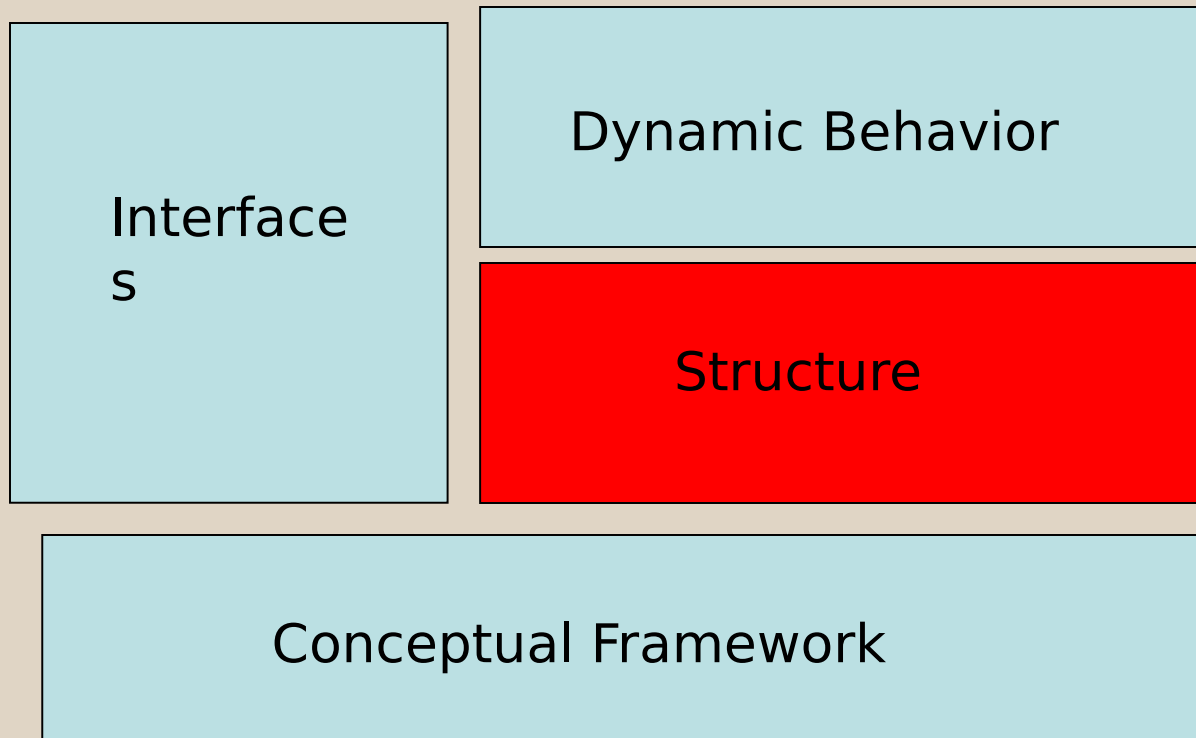
Class Diagram



Class Diagram

Static Structure: things that exist and their relationships, some to become software objects, some not





Requirements
Stage

-Context
Diagrams

Development
Stages

- Software
components

- databases,

-software
entities

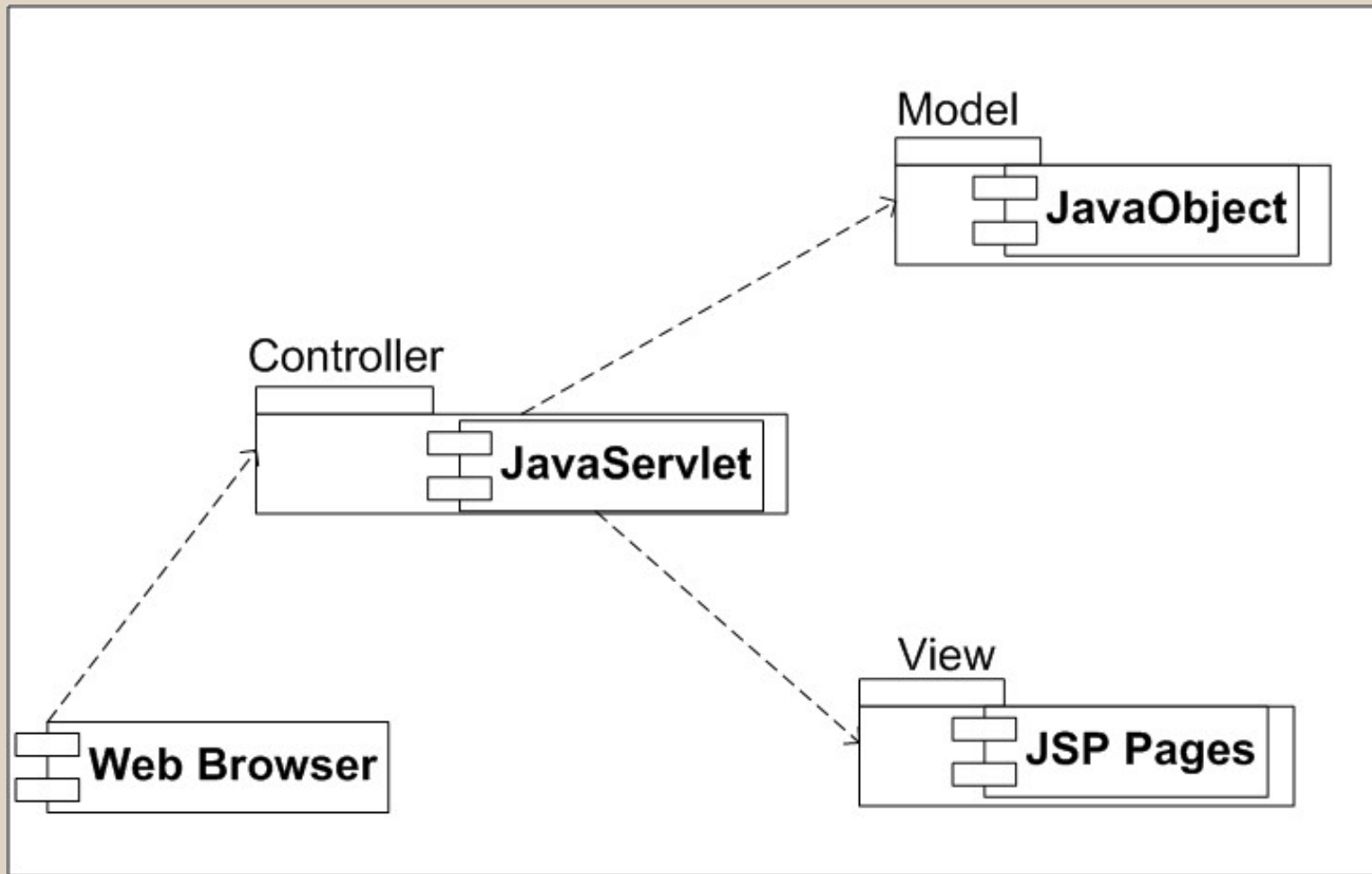
CRC

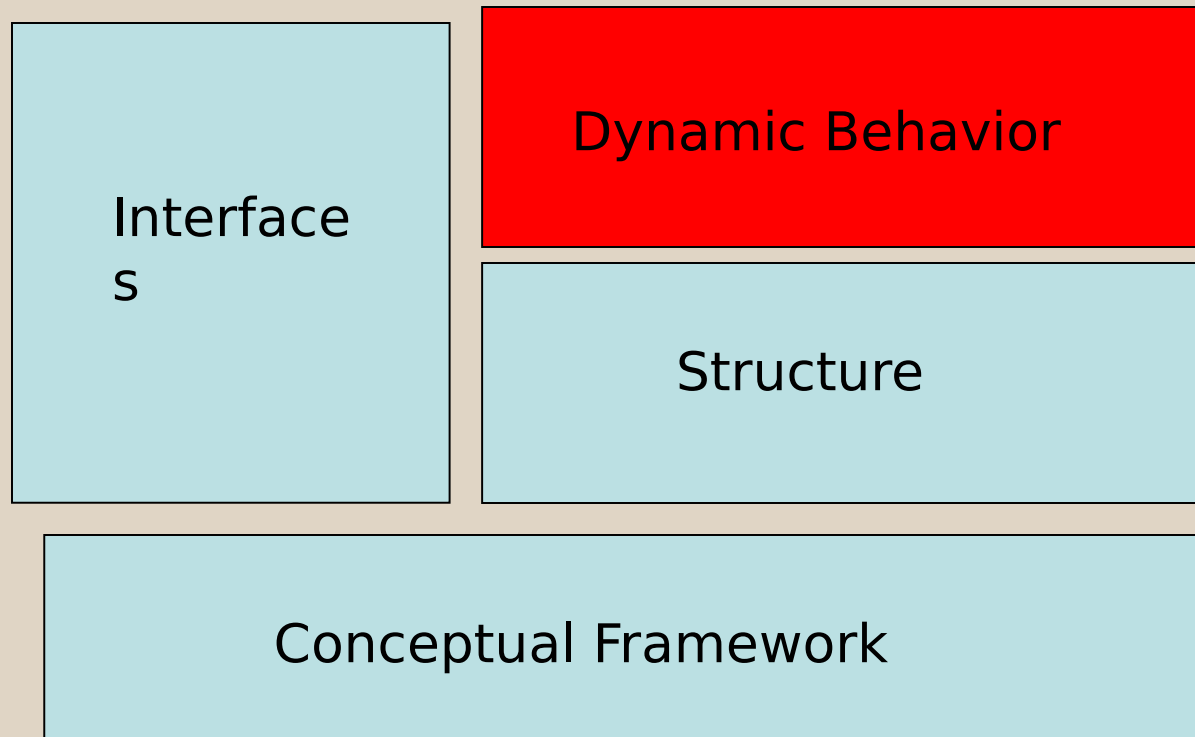
card level

Component Specifications

| | |
|-------------------------|--|
| Component | A Unique Name |
| Responsibilities | Functions and Methods |
| Collaborators | Other Components that must be interfaced with |
| Notes | System constraints on the components, e.g., concurrency and persistence |
| Issues | List of issues remaining to be resolved |

Component structure example

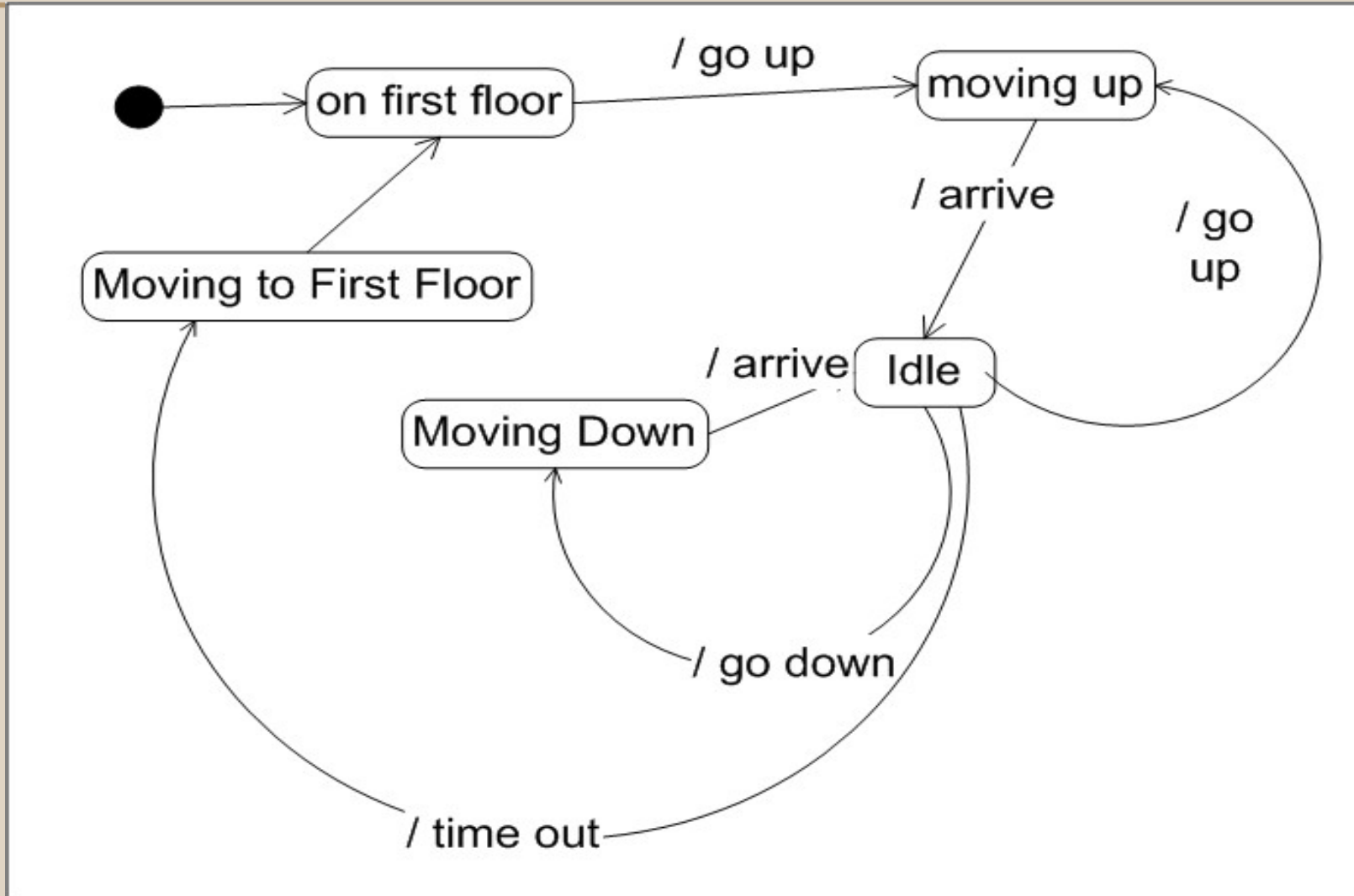




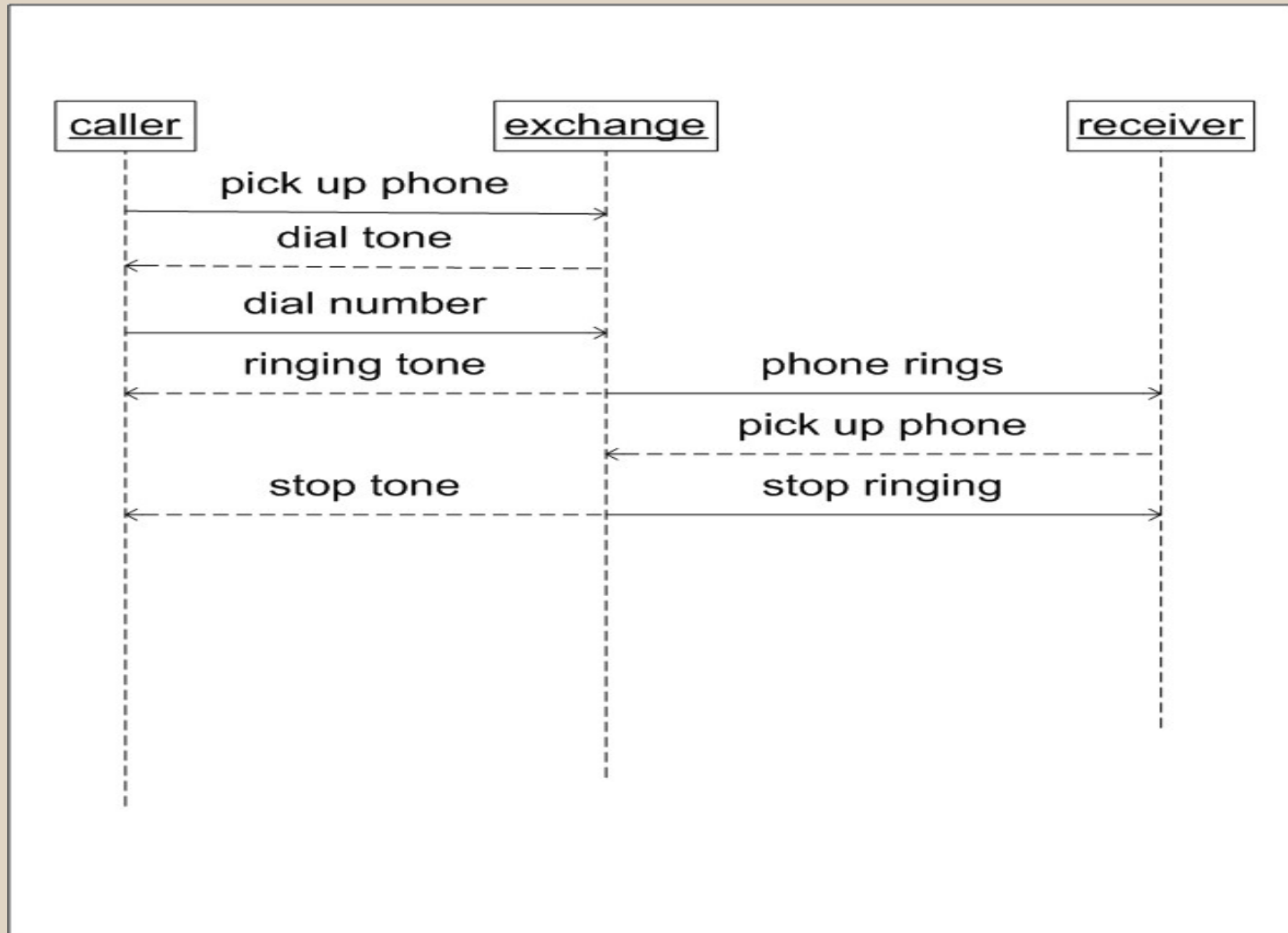
My personal opinion is that trying to document the flow of messages between objects is too hard.

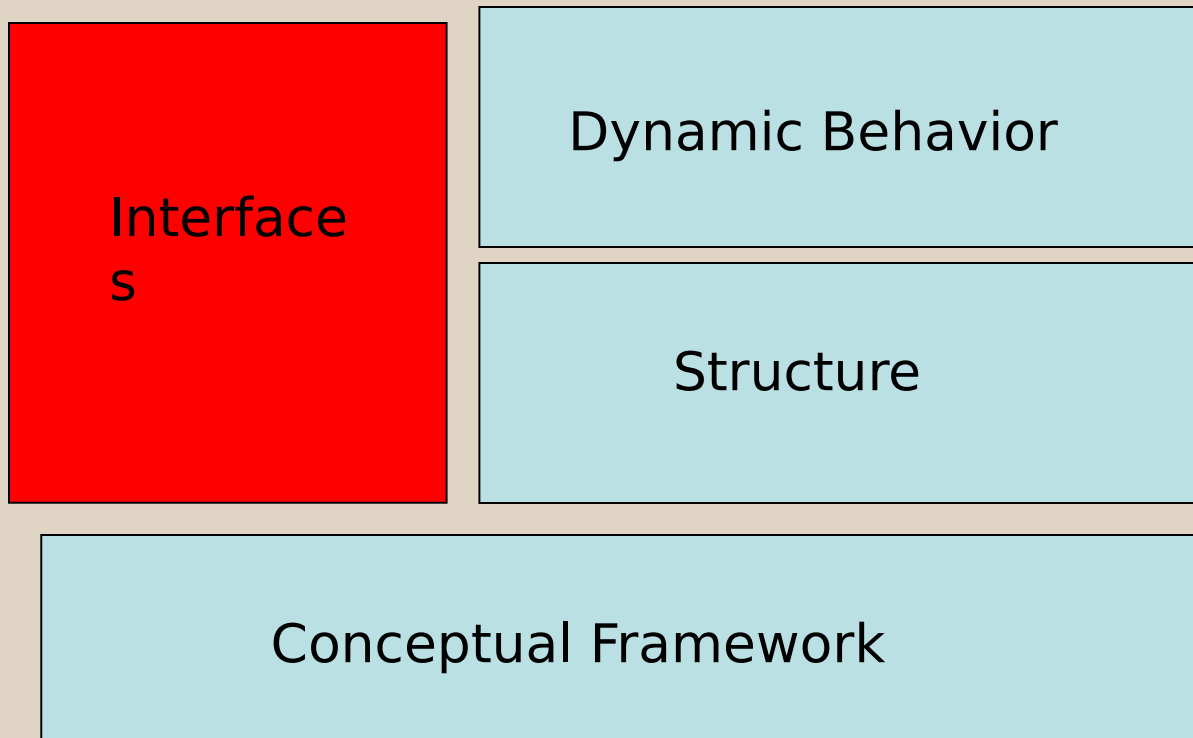
I suggest one stick with activity diagrams at requirements and design stage and state charts as formality is needed

State Charts



Sequence Diagrams





Stick with an
IDL description
for the
interfaces